Indigenous Language Rights & Realities

A section of *LD&C*

Volume
1 (2025)

Pages
72-96

Received
23 Feb. 2024

Handle
https://hdl.handle.net/10125/75046

Online
http://nflrc.hawaii.edu/ilrr

Citation
Vincent, Audra, Alice Kwak, Shannon T. Bischoff, Amy Fountain, & John Ivens. 2025. It is not an archive, and it will never be done: Building the Coeur d'Alene Online Language Resource Center [Indigenous Language Rights & Realities. 1: 72-96]. *Language Documentation & Conservation.*

It is not an archive, and it will never be done: Building the Coeur d'Alene Online Language Resource Center

Audra Vincent
Coeur d'Alene Tribe

Alice Kwak
University of Arizona

Shannon T. Bischoff
Purdue University Fort Wayne

Amy Fountain
University of Arizona

John Ivens
University of Arizona

For language workers in Indigenous communities, language documentation done by outsiders, materials created by previous generations of workers, and even materials created contemporaneously, can all be useful tools for language teachers and learners. Workers may, however, lack appropriate access to these materials, some materials may be stored in vulnerable locations, or in obsolete formats. One approach to these issues can be the development of online language resources that could provide both appropriate access to and safe storage of these valuable materials. In the Coeur d'Alene community, the choice was made in 2009 to begin the process of developing a website that would hold much of the documentation that had previously been held in external collections, on paper and in various now-archaic electronic formats. The current result of that effort can be seen at https://thecolrc.org, which is a web application that is still being expanded and improved with the goals of providing both appropriate access and safe storage to a variety of language materials. In this paper we discuss the partnerships and collaborations that have been necessary to ensure the longevity of this project, the advantages and disadvantages associated with the choices we've made, and the technical infrastructure and resources required to develop this application. We take the position that even though the work focuses on safe storage and appropriate access, there are structural reasons that it is not an archive; and that the decision to use digitization and application development for this purpose means that the work will never be completed.

E-ISSN 1934-5275

1. Introduction[1]

In this paper we present a brief history of the Coeur d'Alene Online Language Resource Center (COLRC), an online Coeur d'Alene (Salish/USA) language resource (initially developed in 2009 with basic html, css, and JavaScript) and recent work transforming this web resource from a simple website to an application that includes a dynamic database.

Our goal is twofold: (1) to share and motivate the strategies we have employed to transition the COLRC from a simple website into an application and (2) to dispel some of the myths of digital archiving that emerged in the early aughts of the 21st century with the ubiquity of the internet, ease of web development, and desire to make language resources easily accessible, and manageable for generations to come.[2] The paper will focus primarily on (1); however, (1) should make it clear why (2) is necessary: internet infrastructure is fragile, becomes obsolete, must be updated, must be replaced, and requires a great deal of resources (human, financial, and material), that may not be readily available or sustainable: In contrast, a book on a shelf in a library is more likely to be around 100 years from now than a website, application, or electronic database and not in need of physical maintenance and upgrading. However, a book may not be easily accessible to all that may be interested in its content[3].

In regards to our second goal, we employ Johnson's (2014, p. 142) definition of an *archive*:

> An archive is a trusted repository created and maintained by an institution with a demonstrated commitment to permanence and the long-term preservation of archived resources.

In a similar vein, Yi et al. (2022, p. 2) define a language archive as follows:

> . . . a repository of language data (broadly construed), such as audio and/or video recordings,transcriptions, and translations, whether in physical or digital format, created with the purpose of preserving and disseminating those materials.

Initially, in 2009 and 2010 in the first phase of the COLRC, the developers of the COLRC had the "full support" of two universities that committed to the permanence and long-term preservation of archived resources in the COLRC. This "commitment", led the developers to assume that crucial server resources would be available, at least for the lifetime of a typical faculty member (20-30 years)[4]. However, in 2013 one of the universities modified their policies regarding server support and ended their relationship with the COLRC. In 2017, the second university modified its policies regarding server access as it transitioned away from managing university wide physical servers for the cloud. At that time, the COLRC was "grandfathered" into a server "program" along with a small number of units allowed to manage resources on university housed, managed, and maintained servers. Since 2017, all but a very few of the grandfathered servers have been migrated to the cloud at the expense of those utilizing them.

At the time of writing, it is no longer the case that the developers of the COLRC have the full support of any university or institutions in regards to assured server access, management, and maintenance. As a result, the developers/authors are now working to find ways to ensure the longevity of the COLRC as a digital resource with stable server support. It is issues such as these, that we believe necessitate an evaluation of what it means for a language archive to have any sense of permanence (beyond the working lifetime of the creators) and to thus truly be an archive: especially at the level of a single language community (but see Yi et al. 2022 for discussion of how similar issues impact large scale language archives).

In short, we believe the term "archive" is problematic in that it suggests a finished static repository with sound and dependable infrastructure and the backing of a well-resourced institution. Instead, language archives, as Henke and

[1] This discussion reports on work supported by the National Science Foundation awards BCS-1160627 and BCS-1160394 and the National Endowment for the Humanities award PD-261031-18, without which the work would not have been possible.

[2] See Henke and Berez-Kroeker (2016) for discussion of a history of digital archiving of Indigenous language resources.

[3] See Yiet al. 2022 for discussion of such issues.

[4] Curiously, Henke and Berez-Kroeker (2016) identify the Endangered Language Fund (ELF) as one of the growing number (at the time) of funding organizations requiring some form of commitment to archiving for all grants. Presently, ELF is modifying this requirement due to the challenges faced by depositors and is no longer collecting materials from awardees due to the challenges of managing a digital archive of such resources at ELF (co-author Bischoff is presently President of ELF).

Berez-Kroeker (2016: 426) note (paraphrasing Albarillo & Theiberger 2009 and Holton 2013), are . . .

> ... an ever-unfinished research product that involves taking in new information, digitizing old materials, and navigating developments in digital infrastructures, formats, and standards.

In the remainder of the paper, we turn to navigating developments in digital infrastructures, formats and standards. We begin with a brief history of the COLRC. We then turn to the process of transitioning the COLRC from a simple website into an application with a dynamic database.

2. The foundation[5]

At the beginning of the 21st century, much work was being undertaken by members of the Coeur d'Alene Community, and by scholars of Coeur d'Alene, to advance the collaboration begun between community scholars and academic scholars throughout the 20th century (cf. Bischoff, Fountain & Vincent 2018 for a discussion). This included the development of significant resources for the teaching of the Coeur d'Alene language by L. Nicodemus and others (Nicodemus 1973, 1975a, 1975b; Nicodemus et al 2000a, 2000b), the advancement of theoretical work on Coeur d'Alene grammar by academic scholars including Ivy Doak (e.g. Doak 1992, 1997; Doak and Mattina 1997; Doak and Montler 2000), Shannon Bischoff (2001, 2007, 2011a, 2011b), Audra Vincent (2014), and the beginning of an effort to render the products of earlier collaborations more readily accessible to both scholars and community members.

By 2009 co-author Bischoff and Yasin Fort, an undergraduate student at University of Puerto Rico Mayagüez, conceived and completed a pilot project to digitize and begin to repatriate materials produced by Coeur d'Alene speakers and scholars Dorthy Nicodemus, Julia Antelope Nicodemus, Lawrence Nicodemus, and Tom Miyal and linguistic anthropologist Gladys Reichard. These materials included what are referred to in the literature as myths and tales (cf. Bischoff, Fountain, & Vincent 2018 for discussion of the resources and the team that created them). The goal was to ensure the material would again be available to the Coeur d'Alene community.

Bischoff and Yasin Fort's work was begun after it had become clear that many of the products of previous collaborations had become inaccessible to the community, and that some were in danger of permanent loss. At the onset of that project, Bischoff reached out to the then Director of Language Programs at the Coeur d'Alene Tribe to identify the goals and needs within the community. Objectives identified by the Director included the development of a searchable online English/Coeur d'Alene dictionary, and the development of digital access to the heritage materials created by the researchers described above. These goals overlapped with academic interests in the emerging field of digital language resources and the project was undertaken to serve both community and scholarly needs. It should be noted that these actions and this project where greatly inspired by the work of Junker described in Junker (2018)[6].

Bischoff and Yasin Fort developed and released the Coeur d'Alene Archive[7] and Online Language Resources (CAOLR)[8] in an attempt to meet both the goals of the Tribal Language Programs and their own academic goals. In one summer, they managed to digitize more than 1,200 pages of original field notes produced by Reichard and the Nicodemus group, collect and secure copyright permissions to incorporate and/or link to earlier scholarly work, and various other previously published but now hard-to-find resources on the language, as listed Table 1 below. All of these materials had been gathered in the CAOLR, and shared with the Coeur d'Alene Language Programs, by the summer of 2009. Remarkably, this work had been done with no external funding, by two enthusiastic volunteers (Bischoff and Yasin Fort) who were relatively new to web development and language archiving, and who had no special training in these areas.

One of the most innovative features of the CAOLR was the set of options users had to view the previously unpublished

[5] This section benefits greatly from work published in Bischoff, Fountain, & Vincent 2018.

[6] In 2005 Bischoff attended a talk by Junker at the Workshop on the Structure of Constituency in Languages of the Americas at the University of Toronto. The talk presented work described in Junker (2018) which planted the seed for the project described.

[7] Here the Bischoff and Yasin Fort saw the COALR as a "repository" supported by the university. Thus, the use of the term "archive".

[8] The original CAOLR remains available at http://lasrv01.pfw.edu/crd_archive/start1.html. It has since been redesigned and re-implemented as the Coeur d'Alene Online Language Resource Center (COLRC).

field notes digitized by Bischoff and Yasin Fort. In addition to full-screen displays of pdf or image (png) files, users could select a split-screen view, illustrated in figure 1 below, showing the evolution of each set of hand-written field notes to typed manuscript versions. Both the top half (the hand-written notes) and the bottom half (the typed manuscripts) can be scrolled through independently, allowing the viewer to match the original and revised transcriptions of the texts. Both the hand-written fieldnotes and typed manuscripts often have editorial marks and hand-written comments, often the work of Reichard providing a glimpse of the collaboration among these scholars that occurred during the transcription and recording of the oral narratives collected. The CAOLR represented an impressive and important achievement – and in 2010 a new phase of the project was undertaken to review and revise the underlying infrastructure of the site, in light of emerging best practices for web-based language resource development, as these were beginning to be developed in, for example, Farrar and Lewis (2007).

Table 1. Primary language resources in the CAOLR

CAOLR Resource	Original Source
Coeur d'Alene/English root dictionary **(searchable)**	**Lyon and** Greene-Wood 2007
Coeur d'Alene /English stem list (searchable)	**Reichard 1939**
Coeur d'Alene /English affix list (searchable)	**Reichard 1938**
Field notes (typed, handwritten, 1,200 pages, 48 narratives, **available as** pdf and png)	**Reichard et** al, unpublished
Published English translations of narratives	**Reichard with** Froelich 1947
Guide to conversions between orthographic conventions **(Reichard's, Salishan,** Coeur d'Alene community **orthography)**	**Various**
Links to other web-accessible public works	**Doak 1997,** Doak and Montler **2006, Reichard** 1938, Reichard **with Froelich** 1947, Teit 1917

The CAOLR provided a place for the storage of digital surrogates of historical documents, but did not in itself meet the definition of a "digital archive" that was emerging at that time (circa 2010). We surveyed the available literature and found two key resources to guide our redevelopment of the CAOLR: Chang's (2010) TAPS Checklist – a resource that provides guidance in evaluating the trustworthiness of online language archives, and Bird and Simons' (2003a, 2003b) work on development of online language archives. Our goals were to:

- Ensure that the site, and the material it contained, was developed in a way that was trustworthy, durable, discoverable, appropriately expandable and sustainable;
- Regularize the site so that it was in line with best practices for modern web development generally, and online language archiving specifically;
- Secure the viability of the site through the near future, and make sure it could be repatriated entirely to the Coeur d'Alene community when the community believed repatriation to be feasible and appropriate; and
- Complete this work in a manner that was as consistent as possible with the "grass roots" model of development used by Bischoff and Yasin Fort (see Bischoff and Fountain 2013 for discussion of this model, and its consequences).

By 2013 this collaboration resulted in a new site, the Coeur d'Alene Online Language Resource Center (COLRC)[9]. The COLRC met the following best practices for web development and online language archiving at the time, in addition to housing all of the content and features originally provided by the CAOLR:

- In all resources in the COLRC, complete and recognizable metadata records are provided in a standard format. We elected to follow the conventions of the Dublin Core Metadata Initiatives[10] (DCMI);
- All resources are stored in such a way that data is separated from presentation – data other than image files are stored and maintained in xml, the site's navigation and presentation are managed via css;
- The site utilizes html 5 standards for presentation, all non-textual resources (audio files, images) are stored in standard and durable formats (pdf, png, jpg, mp3 and wav); and
- The search functionality is expanded to allow simultaneous searching in any of the three supported orthographies across all searchable resources, and renders more resources on the site easily discoverable by users in the Coeur d'Alene and scholarly communities.

More important than these technical advancements, however, was the development of a clear and collaborative Mission Statement for the COLRC[11]. This statement articulates and commits the site to responsible development and management in collaboration with (and with leadership from) the Coeur d'Alene Tribe via the Language Programs office of that Tribe. The development of the Mission Statement required the team to grapple with difficult questions of sustainability, ongoing maintenance, and ongoing relationships among the various stakeholders in its development and use. These issues are difficult because they require the evaluation of long-term goals and requirements in an environment of short-term funding and rich and abiding uncertainty: Which we discovered with issues related to server access described above.

The development of the COLRC was undertaken with the support of the National Science Foundation's Documenting Endangered Languages (DEL) Program,[20] and was therefore subject to demands that the material in the site be accessible to scholars as well as community members (see Benedicto 2018 for discussion of this issue). The history of work on Coeur d'Alene language has always been marked by collaboration between these two groups, however – we view the development, maintenance, and improvements of the COLRC as just another step in that history of collaboration.

In the sections that follow, we relate in some detail the process of creating the COLRC from the point of view of the developers. Our intention is to provide enough information to support readers who may wish to embark on similar projects, to de-mystify some of the technologies involved, and to provide an inventory of the kinds of skills and resources we've used in the process.

3. In the beginning: A simple website

Every website that can be accessed is really a directory of files that live on a computer that is connected to the web, and that is set up as a 'web server'. A web server is a combination of hardware and software that work together to (1) store files and (2) display them via a web address (or 'URL'). Most modern desktop and laptop computers can function as web servers if you want them to; but most websites need to be 'hosted' on a web server that is set up to be always on, to be able to understand and use all the different kinds of files that you want to use to make your website, that has some set of humans devoted to making sure it doesn't break or get confused, that its software gets regularly updated, and that it has back-up and fail-over protections built in. Production-level web servers are relatively costly to run, but many organizations - such as schools, libraries, Tribes, and archives, run them and might be willing to host your website. Individuals and groups can also pay companies like Amazon, Google, and the like for space on web servers. So, when it was determined that the Coeur d'Alene Language Programs wanted to get access to some language documentation materials via a new website, linguist Shannon Bischoff worked with a University where he worked to get access to a web server,

[9] http://lasrv01.pfw.edu/COLRC/

[10] DCMI standards and guidelines can be found at http://dublincore.org/.

[11] http://lasrv01.pfw.edu/COLRC/home/mission.php

and he then got to work.

In 2009, with assistance from one undergraduate independent study student, Musa Yassin Fort, Bischoff embarked on a project that would provide web-based access to Coeur d'Alene documentary materials; formatted as a root dictionary, a list of stems, a list of affixes and a set of more than 1,200 pages of typed and handwritten original field notes. The site created at that time, currently available at http://lasrv01.pfw.edu/crd_archive/start1.html, is a 'simple website'. By this name we don't mean that it was trivial to create or launch, but instead that it is built on a set of technologies that were and are accessible to individuals with very little technical knowledge (see Bischoff & Fountain 2013 for details of the process and technology used at the time).

A simple website is a collection of presentation files, content elements, and scripts. Presentation files are responsible for setting up the formatting of web pages, providing spots for navigational elements and page content. These are typically written in hypertext markup language (html), a simple language that can be easily typed in a plain text editor such as Microsoft's NotePad or Apple's TextEdit**.** Example (1) is very simple html file, note that it uses 'tags' that are enclosed in angle brackets to define spans or areas in the web page; every 'tag' has an opening element and a closing element, the closing element matches the opening element exactly except that the tag has a leading '/':

(1) Simple HTML code.

```
<html>

<h1> Hello World!</h1>

<p>This is my web page</p>

</html>
```

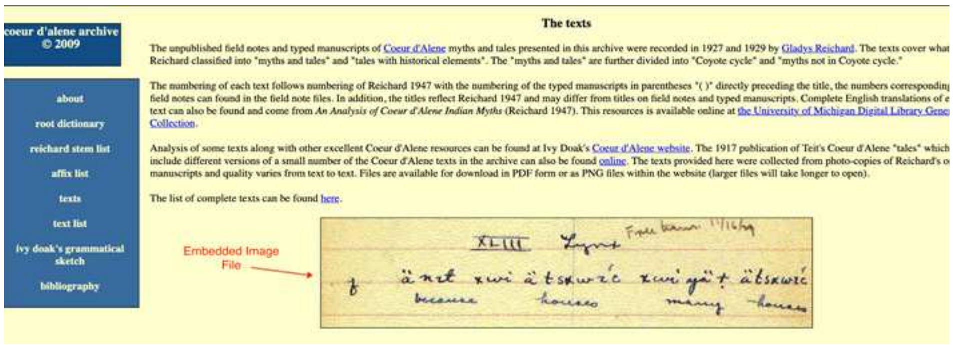
Content elements are the things that the page is supposed to display to a viewer. Content elements can be images, links, tables of data, sound or video files, paragraphs of text, navigation menus, etc. Content elements can be included in a programmatic file by link or embedding.

An example of a presentation file that displays a content element (an image file called 'image.png') is given in (2).

(2) Simple HTML code that displays a content element.

```
<html>

<h1> Hello World!</h1>

<p>Here is an image taken from Reichard's original field notes:</p>

<center><img_src="image.png"></center>

</html>
```

An example of an embedded image file is shown in Figure 1 below:


coeur d'alene archive
© 2009

about
root dictionary
reichard stem list
affix list
texts
text list
ivy doak's grammatical sketch
bibliography

The texts

The unpublished field notes and typed manuscripts of Coeur d'Alene myths and tales presented in this archive were recorded in 1927 and 1929 by Gladys Reichard. The texts cover what Reichard classified into "myths and tales" and "tales with historical elements". The "myths and tales" are further divided into "Coyote cycle" and "myths not in Coyote cycle."

The numbering of each text follows numbering of Reichard 1947 with the numbering of the typed manuscripts in parentheses "( )" directly preceding the title, the numbers corresponding field notes can found in the field note files. In addition, the titles reflect Reichard 1947 and may differ from titles on field notes and typed manuscripts. Complete English translations of e text can also be found and come from *An Analysis of Coeur d'Alene Indian Myths* (Reichard 1947). This resources is available online at the University of Michigan Digital Library Gene Collection.

Analysis of some texts along with other excellent Coeur d'Alene resources can be found at Ivy Doak's Coeur d'Alene website. The 1917 publication of Teit's Coeur d'Alene "tales" which include different versions of a small number of the Coeur d'Alene texts in the archive can also be found online. The texts provided here were collected from photo-copies of Reichard's o manuscripts and quality varies from text to text. Files are available for download in PDF form or as PNG files within the website (larger files will take longer to open).

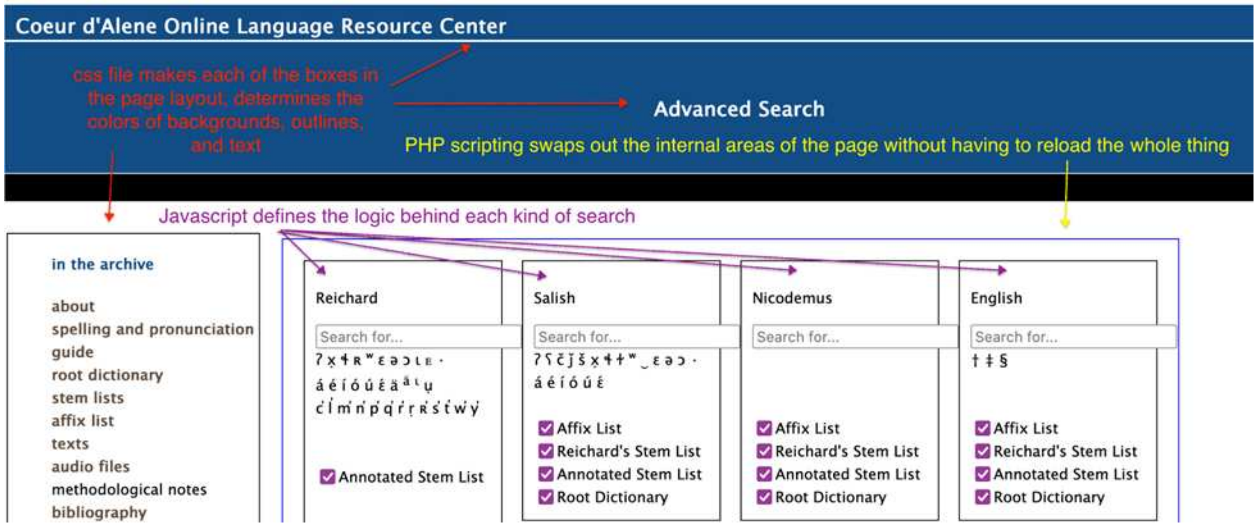
The list of complete texts can be found here.

Figure 1. Embedded image in a simple web page

Content elements may also be blocks of html, for example those that encode headings, paragraphs of text, tables, lists, and the like. In the preceding examples, the tags <h1></h1> wrap text that is meant to appear as a top-level heading, <p></p> wrap a paragraph of text, and <center></center> tell the page to present the embedded image horizontally centered on the web page.

Scripts are snippets of some kind of programming language - usually CSS, JavaScript or PHP - that allow the web designer to create a consistent appearance of elements in multiple pages (CSS, or 'cascading style sheets'), build in some basic kinds of interactivity (JavaScript) or dynamically build a page based on some simple instructions (PHP).

In 2013, the Coeur d'Alene site was redeveloped with attention to best practices in community-based online archiving (Chang 2010), but still as a simple website; currently available at http://lasrv01.pfw.edu/COLRC/. This version of the site incorporates scripts of all three types - a CSS script to provide consistent styling to all the pages, search functions defined in JavaScript, and PHP so that the same 'frame' can be maintained for each page, with different sub-pages popped into the central panel.


Coeur d'Alene Online Language Resource Center

css file makes each of the boxes in the page layout, determines the colors of backgrounds, outlines, and text

PHP scripting swaps out the internal areas of the page without having to reload the whole thing

Javascript defines the logic behind each kind of search

Figure 2. A simple web page that uses CSS, JavaScript and PHP scripts

In simple websites, even if we are displaying data, the data are stored either directly in the html content, or in additional text or text-like files[12]. In the 2013 version of our site, dictionary-like data are stored in plain text files, also sometimes called 'flat files', with columns separated by strings of three colons; so, the actual data that appear as entries under the root √bc are stored like this:

(3) Rows from a flat file containing material for the Coeur d'Alene root √bc

bc:::1:::buc:::buts:::† boots. (n.)

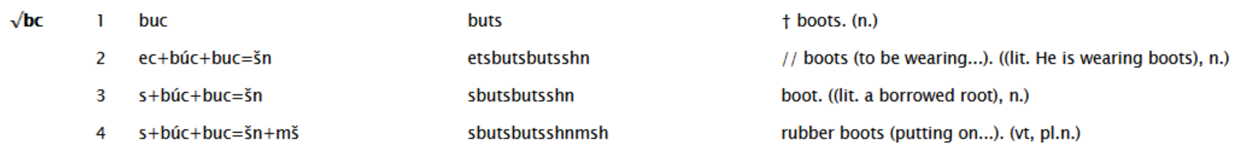
bc:::2:::ec+búc+buc=šn:::etsbutsbutsshn:::// boots (to be wearing...). ((lit. He is wearing boots), n.)

bc:::3:::s+búc+buc=šn:::sbutsbutsshn:::boot. ((lit. a borrowed root), n.)

bc:::4:::s+búc+buc=šn+mš:::sbutsbutsshnmsh:::rubber boots (putting on...). (vt, pl.n.)

These rows are displayed on the web page like this, by means of a tiny JavaScript function that can read the text file and put each element into the correct material into the correct column in an html table:

(4) Rows displayed on the web page



√bc	1	buc	buts	† boots. (n.)
	2	ec+búc+buc=šn	etsbutsbutsshn	// boots (to be wearing...). ((lit. He is wearing boots), n.)
	3	s+búc+buc=šn	sbutsbutsshn	boot. ((lit. a borrowed root), n.)
	4	s+búc+buc=šn+mš	sbutsbutsshnmsh	rubber boots (putting on...). (vt, pl.n.)

Finally, simple websites are really collections of one or more different files, and when you visit a simple website and move between the pages of that site, you're really viewing different files that are all stored in one or more 'folders' or 'directories'. You can tell where you are in the site because the URL shows you the file structure of the site. For example, if you visit this page in our 2013 site: http://lasrv01.pfw.edu/COLRC/dictionary/, you are viewing a file that lives in a folder or directory called 'dictionary', and that folder lives in a bigger directory or folder called 'COLRC'. If you then click through to the 'stem lists', http://lasrv01.pfw.edu/COLRC/stem_list/, the server stops showing you the file from 'dictionary' folder and instead shows you a file from the 'stem_list' folder. The directory structure for this site looks something like this[13], with the files ending in .html being the presentation files for each area of the site:

(5) The directory structure of a simple website

COLRC

- dictionary
  - dictionary.html (= http://lasrv01.pfw.edu/COLRC/dictionary)
- stem_list
  - stem_list.html (= http://lasrv01.pfw.edu/COLRC/stem_list)
- affix_list
  - affix_list.html (= http://lasrv01.pfw.edu/COLRC/affix_list)

[12] Best practice would require that these files be prepared using 'extensible markup language', or 'xml'; xml files are constructed just like html files, except creators make up their own custom tags, and provide a template that describes the tags, their meaning, and the files' internal organization.

[13] This is simplified for clarity purposes; but we hope it makes sense.

```
📁texts
        📄text1.html   (= http://lasrv01.pfw.edu/COLRC/texts/text1.hml)
        📄text2.html   (= http://lasrv01.pfw.edu/COLRC/texts/text2.hml)
        📄text3.html   (= http://lasrv01.pfw.edu/COLRC/texts/text2.hml)
```

In sum, simple websites live on web servers, are built of mostly hand-writable, human-readable, easy-to-learn parts, and allow viewers to see different files that are organized into different folders. They can be very labor-intensive to build, of course, and require careful digitization and organization of the materials displayed on the site, but they are very robust and can last a long time. There are simple websites that were developed in the very first years of the world wide web that are still functional today (you may wish to visit https://spork.org/, which has apparently been up and running unchanged since 1996)[14].

However, simple websites also have drawbacks that make them unacceptable options for many language communities. First, they are difficult to secure and protect; while there are some low-tech ways of hiding pages from the general public (e.g., with very simple unencrypted log in links), these are easy to defeat and hard to maintain. Second, they do not offer true interactivity; truly powerful and flexible searches and sorts require much more complicated programming; Third, they can only be updated by the people who build and maintain them. It's not possible in a simple website to update the data without also rewriting the code.

For these reasons, a third version of the Coeur d'Alene application was developed beginning in 2018 - and we are continuing to build enhancements and extensions. We move on now to describe the rationale for moving from a simple site to an application, and then describe the technical architecture and skill sets necessary for modern web application development, in our experience.

4. For added functionality: A web application

Beginning in 2018, the Coeur d'Alene team started working to convert our simple website (http://lasrv01.pfw.edu/COLRC/) into a web application. The goals of this effort, which we refer to as 'the COLRC 2.0', included adding functionality to the site that would allow the language program manager to add and edit material on the site, expand the kinds of materials that would be available, secure the resources to allow some resources to be available only under log in, and that would provide more flexible and dynamic site-wide search and sort functionality.

These goals meant that we needed to move from flat files to a database so that (among other things) any changes would be characterized by transactional integrity; that we needed to create user authentication and authorization to ensure that only legitimate users had access to edit or update data; and that we needed to create a much more flexible and interactive interface for visitors. The difference in skills and resources needed for the COLRC 2.0 compared to our previous efforts has been striking; but so have the rewards of building a more advanced application.

Web applications are different from simple sites in a number of ways. To get a feel for some of these differences, compare the sites we've discussed so far with a site like https://www.amazon.com/. In addition to just having an almost infinite number of 'pages', Amazon looks different to each individual who uses it. It has a login feature, and it remembers information from past visits. It changes constantly - and not because Amazon engineers are constantly re-writing the web pages. While a community language resource web application certainly doesn't need to be as massive and fancy as Amazon, we do want some of those core features. So, what are the pieces and parts necessary to make that happen?

First, web applications are written almost entirely in computer languages - most prominently JavaScript. We used little snippets of JavaScript in our simple site, but in developing a web application we need to learn that language well

[14] It is also possible to visit the Internet Archive's "Wayback Machine" https://web.archive.org/ to view early html websites to see how they have changed over the years.

enough to make almost everything. When you look at a simple website in a web browser like Google Chrome, Apple Safari, or Mozilla Firefox, your browser is 'reading' the html that you wrote yourself. However, when you look at a web application, the browser is able to understand and use JavaScript directly. One way to think of this is that JavaScript is every browser's first language, so it's super-fast and efficient for browsers to work with.

Here is an example of a very simple web page, written in JavaScript[15]:

(6) A simple web page, written in JavaScript

```
function MyGreeting() {
  return (
        <span>
        I'm happy to meet you
        </span>
  );
}
export default function MyApp() {
  return (
        <div>
        <h1>HelloWorld</h1>
        <MyGreeting />
        </div>
  );
}
```

Second, web applications typically require layers of software in order to work. In our case, the layers include a database, a back-end, some middleware, and a front-end. Finally, the application is containerized[16]. In the next subsections, we discuss the reasons that each of these layers is needed, and the particular applications we chose to use. Our hope is that this information will be of use to others who might want to work on web application development in their communities. For each layer, we discuss what the software does, why it is necessary, and then share the particular versions we selected for the COLRC 2.0.

We provide complete technical schematics for all components of the COLRC 2.0 in Appendix A. We hope that these are sufficient to allow a software engineer to understand the system architecture and make informed decisions about how to develop a similar application. In the remaining subsections here, we discuss key components of that architecture, in the hopes that non-engineers will come away with a basic conceptual understanding of the different pieces and parts of a web application like ours.

[15] Specifically, 'React'-type javascript. Javascript comes in a variety of ... dialects? Our project chose this one, you can find out more about it at https://react.dev.

[16] See Containerization below for discussion.

5. A relational database

In this section, we discuss the basic structure and functions of a traditional relational database that relies on Structured Query Language (SQL) to store and retrieve information. Relational databases and SQL were both introduced in 1970 (Date 1984; Codd 1970), and are among the oldest and most stable kinds of computer applications.

What does it do?

Databases store data in the form of tables and relationships between tables. For example, I might have a table (converted from example (3)) to store the roots I want to display in a root dictionary

(7) A partial roots table

id	**root**	**sense**	**orthography 1**	**orthography 2**	**english**
1	bc	1	buc	buts	† boots. (n.)
2	bc	2	ec+búc+buc=šn	etsbutsbutsshn	// boots (to be wearing...). ((lit. He is wearing boots), n.)
3	bc	3	s+búc+buc=šn	sbutsbutsshn	((lit. a borrowed root), n.)
4	bc	4	s+búc+buc=šn+mš	sbutsbutsshnmsh	rubber boots (putting on...). (vt, pl.n.)

Every row in the table has an ID, and that ID is a unique number for that row. We will call that unique ID the 'primary key' for the table, because I can always find exactly one entry in the table if that matches this ID.

Let's say I also want to store information about who entered each root in the roots table:

(8) A users table[17]

id	**first**	**last**	**username**	**email**
1	Lawrence	Nicodemus	original	original@domain.org
2	Audra	Vincent	manager	audra@domain.org
3	Cheffrey	Sailto	instructor	cheffrey@domain.org
4	Amy	Fountain	tech	amy@domain.org

Each user has a unique ID, which is the primary key of this table.

Now I am ready to add a relationship to my database, by adding one more column to my roots table - the ID for the user who added that entry. The userId is a primary key from a different table, so in the roots table it becomes a 'foreign key' (9).

We can now describe the roots table and the users table as 'joined', because they're connected by a primary key to foreign key relation. As I build the database, I can build different tables and join them, which chains together lots of kinds

[17] We are not using real email information here.

of information and allows the computer to retrieve it all in a very efficient way. For example, I can display information from both tables in one grid when I show the dictionary entries online; I can select which fields from each of the two joined tables I'd like to display, as in example (10).

(9) A partial roots table with a foreign key

id	**first**	**last**	**username**	**email**
1	Lawrence	Nicodemus	original	original@domain.org
2	Audra	Vincent	manager	audra@domain.org
3	Cheffrey	Sailto	instructor	cheffrey@domain.org
4	Amy	Fountain	tech	amy@domain.org

(10) A display that pulls selected fields from two joined tables

root	**orthography 1**	**orthography 2**	**english**	**entered by**
bc	buc	buts	† boots. (n.)	original
bc	ec+búc+buc=šn	etsbutsbutsshn	// boots (to be wearing...). ((lit. He is wearing boots), n.)	tech
bc	s+búc+buc=šn	sbutsbutsshn	((lit. a borrowed root), n.)	teacher
bc	s+búc+buc=šn+mš	sbutsbutsshnmsh	rubber boots (putting on...). (vt, pl.n.)	manager

Why is it needed?

In addition to tables and joins, relational databases also do several things automatically: they produce primary keys that are guaranteed to be unique, they ensure that only the 'correct' data types can be added to a column in your table (some columns can only contain numbers, others text, others date or time stamps, etc.) they log dates and times of changes made to them, and they enforce 'transactional integrity'. This means that they are built to ensure that every time anyone tries to change the data in the database, the change either succeeds or fails 100%, never partially. Let's walk through a simple example that illustrates why transactional integrity is so important.

Imagine that you want to add a new word to your online dictionary. That new word would be added as a new row into the 'roots' table in your database - so it will need to have data required for all the columns in your roots table, including your userId. Let's imagine that you are user 4 in this example. This means that the row you'd like to add needs these values (you don't need an id value because the computer's going to add that automatically):

(11) A row to be added

root	**sense**	**orthography 1**	**orthography 2**	**english**	**userId**
bm	1	s+bam+p	sbamp	speeding	4

Now let's imagine that you push the magic 'insert' button and all of a sudden you have a power outage! What the computer got before the outage was just a partial set of fields, like this:

(12) A row with missing information

root	**sense**	**orthography 1**	**orthography 2**	**english**	**userId**
bm	1	s+bam+p	sbamp	spe	

If you are adding data to a simple flat file, your file is going to have a partial line in it somewhere. That partial line is going to cause your display to break, because all of a sudden, the table you're trying to display doesn't have enough information to fill all the columns. When you refresh your display, you might get an error - or you might get a table that has data in the wrong places, or that's missing crucial information - and you might not even notice the error until much later (when it's going to be very difficult to resolve).

If you're adding the data to a database, the system won't accept that partial entry. It will roll the transaction back, and your system is safe. You can re-do the insert when the power is back, and all is well.

The above is a pretty unlikely scenario, it's much more common that we're doing bigger, more complicated changes on our databases when we change them – for example we might be reading in lots of rows of data, or adding data to multiple tables in one transaction, or there might be multiple people doing work on the database at once. In these circumstances, transactional integrity is absolutely crucial to our ability to maintain our data accurately and safely.

In addition to transactional integrity, relational databases allow us to create flexible, customized displays for data from multiple tables by understanding and executing 'queries'. Queries are structured expressions, usually written in a language called 'Structured Query Language' or SQL, that tell the computer exactly which fields in which tables to return to us and let us see. These expressions, called SQL statements, can be very simple or very, very complicated. Here's a simple SQL statement that asks the computer how many records I have in my 'roots' table:

(13) An SQL statement

`SELECT COUNT(*) FROM roots;`

Database response: 7690

In (13), the SQL statement uses the verb 'SELECT', which means something like 'find', and the term 'count', which means that we want an answer that's the number of items that meet our criteria. The (*) means 'everything', and the modifier 'FROM' tells the machine which table to look at.

(14) is a more complicated SQL statement, that asks for data from multiple joined tables in the COLRC 2.0

(14) A more complicated SQL statement

`SELECT value, english, salish FROM stems LEFT JOIN stem_categories ON stems.category=stem_categories.id WHERE english like '%horse%' order by value, english;`

This statement asks the machine to find the value, english, and salish columns from a table called 'stems', which is joined with a different table called 'stem_categories', using the 'category' field in the stems table as the primary key, and the 'id' field in the stem_categories table as the foreign key; paying attention ONLY to the rows in the stem table that contain the letters 'horse' as a string, even if there are other letters before or after 'horse' (that's what the '%' says), and to sort these alphabetically by the English translation. This query returns the following results:

To a person using our application, a query like (14) is what the machine needs to see in order to process a 'search' that you might run if you wanted to find all the stems in the database that talk about horses. Note that the fifth result isn't about horses at all, but it has the string 'horse' in it, in the compound word 'horseshoe curve'.

If you've gotten this far, you might be feeling disheartened by the complexity of the SQL language. But never fear, we have tools that allow us to create these complicated queries in a more accessible and manageable way, these are discussed in the 'middleware' section below.

If you're interested in building a web application for your language material, though, you might want to invest a little time and interest into learning some very basic SQL.

(15) The result of the statement in (14) above.

value	english	salish
nouns	horse collar, necklace	s-q'ɛl'-ɛps
nouns	horse, pet, domestic animal	ɛsčíčɛʔ
verbs	be black (of horses)	q'ʷəd(u-)
verbs	be well-shaped (as workhorse)	mɛč'
verbs	dive from land (face in horseshoe curve)	us-əlš
verbs	gray (as horse)	x̣iy

(6 rows)

Which database(s) did we choose?

There are many freely available databases to choose from, we elected to use Postgres (https://www.postgresql.org/) based on its reputation for stability and efficiency. Other commonly used relational databases at the time of writing include MongoDB, MariaDB, MySQL, and Oracle. Some relational databases are fine-tuned to store whole documents and to not rely on SQL (these are referred to as no 'SQL' databases), rather than to store information in lots of small, structured tables like ours. As it turns out, we are in the process of adding another database to the COLRC 2.0, and it is a document-based system that will allow us to build searches that can go into various full-text documents. But that is a topic for a future paper.

A back-end

The 'back-end' of an application refers to different services that the computer needs to run for the application to work, but that the person who accesses the application does not see. In our case, we needed back-end services to (1) handle user logins and session security and (2) tell our other programs how to understand our database through 'object relational mapping', which translates between the database's language (SQL) and other programming languages that are used to talk with it (Abba 2022).

What does it do?

The back-end service in our application runs a program that listens for people trying to log in to the application, this is called an 'authorization server'. Not everyone who uses the application will log in, but people who have the authorization to edit data must log in to an account that has been authorized by the language program manager.

When a person logs into the application, they provide their credentials (an email and password) in a form. The back-end service does several steps:

- it encrypts the credentials so that they can't be stolen by sneaky people on the internet; and then
- it checks to see if the credentials match an authorized account in the database 'users' table;

  and then if they match,

    - it finds out what the authorization level is for that account - what kinds of actions the person with that account is actually allowed to do in the system; and then
    - it asks an external system to give back a big, scary encrypted string of numbers and letters called a 'Json web token' or JWT (https://jwt.io/). The token encrypts the users' identity and authorization level, and the token is stored in a special area of the browser called 'local memory'. Every time the person tries to do an action that's limited only to authorized persons, the token is compared to an encryption key that's stored in our application so that it can record who did the action. Local memory
- or if they do not match, it sends an error message back to the person indicating that the login failed.

In addition to managing logins and tokens, the back-end service sets up the database tables in a way that can allow us to (re-)build the database any time we need to.

Why is it needed?

The back-end is needed because it provides these necessary utilities to the application. It's not very complicated, but it's very important.

Which systems did we choose?

The back-end in our system is built using Node.js (https://nodejs.org/). Node.js uses the JavaScript computer language and allows it to run directly on a server, rather than in a web browser. We chose Node for our back-end because we knew we had to learn to use JavaScript for other parts of the system, so being able to use it for the back-end meant we at least would not have to learn another computer language! Other languages commonly used to build back-end services include PHP, Java, Python, among many others.

A router

Multi-part applications need all the parts to be able to talk to and understand each other, and they need rules about how and where each piece talks to the others. For example, if a user wants to log in or log out, those requests need to be sent to a back-end service (see the section 'titled back-end' above), but if they want to add or modify data they need to be able to talk to the relational database. For all this communication, the application runs on a server that manages routing of requests.

What does it do?

A router provides a map of different directories and other components so that everything in the application can talk to everything else, and can send information from one part of the application to another. The router is what stores and delivers pdf and image files to the front-end, and directs log in and log out requests to the back-end.

A router also contains settings that tell the application to run as 'https', rather than 'http'. You might be familiar with the letters 'http' that begin the URL of any website you visit - they stand for 'hypertext transfer protocol'. This is the set of rules and standards that machines understand when they transfer data around the internet. When the 's' is added, the abbreviation means 'hypertext transfer protocol secure', and refers to a special set of rules for moving data around the internet safely, by keeping that data encrypted. Encryption during all sending and receiving of information on a website helps ensure that nobody can sneak into your session and - for example - steal your token and our decryption key!

To use 'https', the system has to be set up in a very particular way, and most of this work is done within the router application. Your URL - in our case 'thecolrc.org' - has to be registered with a domain naming authority (some of these are Amazon Web Services Route 53, GoDaddy, Cloudflare DNS, and IONOS) - this is sort of like registering a physical address of your business with a licensing company. Domain name registration ensures that your URL is only going to be used by you, and won't be copied out from under you and used by sneaky people on the internet.

Once your domain name is registered, you tell the registration system the identifying information (the IP address) of the server where your application lives. Then you initiate a process by which the domain name service does an automatic check on that address to ensure that it really exists and includes proper identifying information. When that check works, the process produces a 'security certificate' that's stored on your server in a special place. When you have that valid certificate in place, you can use the 'https' protocol and all of the information that travels to and from your application over the internet will be safely encrypted. Certificate checks are done periodically - currently the standard is every 30 days - and so the router can be configured to automatically start the certification process every so often.

Why is it needed?

A router is needed because the application has many components, and the components need to know about each other. The front-end needs to talk to the back-end to allow users to log in or create accounts; the back-end needs to talk to the database to allow authorized users to make changes to the data; the database needs to talk to the middleware to get requests for data and return the right data back so it can be passed to the front-end, etc. A router is also needed to ensure that you can run your application using https rather than http.

You may be thinking about times when you have tried to go to a website and gotten a warning message that says something like 'this site's security certificate is expired - it is dangerous to proceed. Do you want to go there anyway?'. Now you know what causes that - either the people who administer the application have not requested a new certificate in time, or the automatic process failed for some reason and they don't realize it yet. It is also possible that the domain name's registration expired - keeping your domain name registered requires payment of an annual fee. The fee can be very small (for domain names that nobody else wants) or quite expensive (for domain names that a lot of people might want). There are folks out there who make money by investing in domain names and trying to resell them at a profit, or who buy them in order to keep them safe (for example, you might look into purchasing a domain like 'yourname.com' if you are thinking of going into politics, so that your opposition cannot buy it and post terrible things about you there), or to hold them hostage.

Which systems did we choose?

We chose a server application called nginx, pronounced 'engine x' (https://www.nginx.com/) to use as our router. Most applications use either nginx or apache[18] (https://httpd.apache.org/) for this purpose. We selected LetsEncrypt (https://letsencrypt.org/) for our certificate management process, it is by far the most commonly used system for certificates today.

Some Middleware

Middleware is software that helps glue the pieces of your application together. Middleware helps make writing and maintaining your application easier, and keeps everything running smoothly. Similar to the back-end software, middleware is not something users ever see, but they benefit from it indirectly.

What does it do?

Remember our examples of SQL above? Middleware is used to provide a friendlier and more efficient way for us to ask our database for data, and then return the requested data in a form that our front-end components can understand. Let's work through an example to help make this clear. Remember (14)? It's repeated here as (14') for clarity:

(14') A more complicated SQL statement

`SELECT value, english, salish FROM stems LEFT JOIN stem_categories ON stems.category=stem_categories.id WHERE english like '%horse%' order by value, english;`

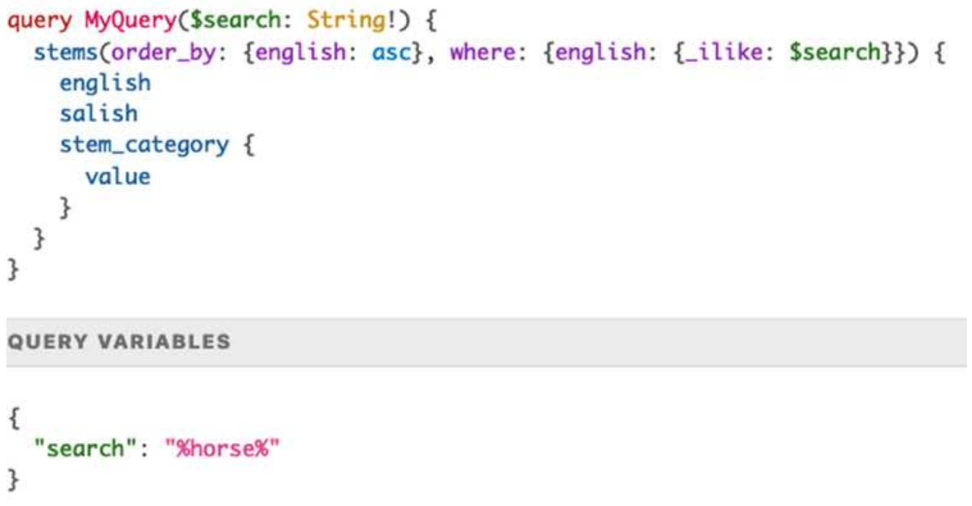
This statement is meant to go to the database and give us back a list of stems whose English translation includes 'horse', and show us three things about each stem - its value, its spelling in Salish, and its English translation. And it's supposed to return these results sorted alphabetically by the English translation. Remember that the 'value' element is stored in one table, but all the other information about the stem is stored in a different table. Instead of trying to construct SQL statements like this one for each kind of query we might need on our site, we use an application called Hasura (https://hasura.io/) which is built on a language called Graph Query Language (GQL, https://www.gqlstandards.org/). GQL was created by engineers at Facebook to improve performance of the Facebook mobile applications, and has recently grown

[18] Isn't it awful that a bunch of tech people were allowed to use 'Apache' as the name for their software and make a bunch of money off of that? Shouldn't they at least have to pay the Apache people for use of this name?

into an internet standard for data management and transfer (Byron 2015; GQL Standards Committee).

Here's an example of how we can use GQL and Hasura to write a request to the database to return data just like in (14), except we can express "%horse%" as a variable - something that we can swap out for other values really easily, depending on what a user would like to search for.

(16) A query in Graph Query Language



```
query MyQuery($search: String!) {
  stems(order_by: {english: asc}, where: {english: {_ilike: $search}}) {
    english
    salish
    stem_category {
      value
    }
  }
}
```

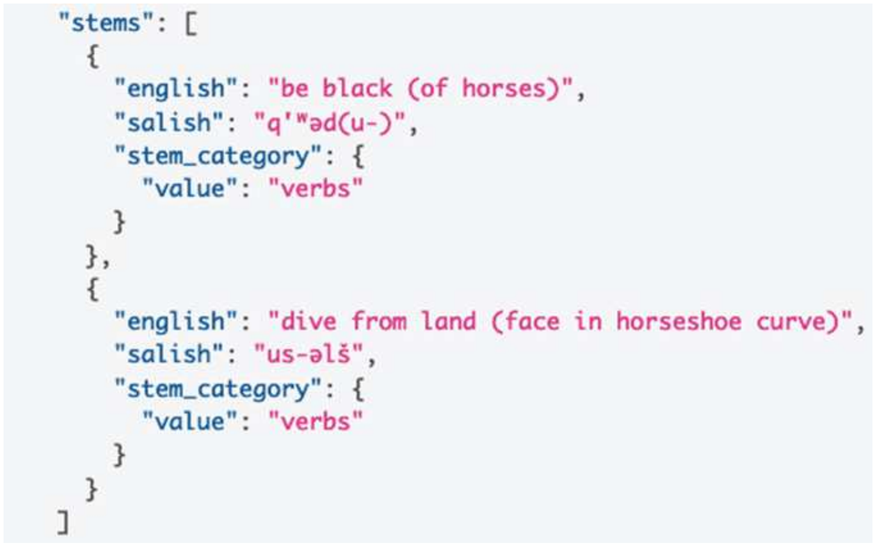
QUERY VARIABLES

```
{
  "search": "%horse%"
}
```

The query returns results that are organized into key-value pairs that live inside of different kinds of nested brackets (technically, the results are returned as an array […] of hashes { … }), and that can be easily fed into a table display in the front-end. Here we have limited the results by searching for 'horses', to save space - only two results are found, and you can see that each gives us the English, Salish, and Stem Category value, and they're ordered alphabetically by the English.

While a normal human might not see much difference in terms of ease of use between the SQL and the GQL statements, to a computer program written in a language like JavaScript, the GQL is far easier and more manageable.

(17) Results of the query in (16)



```
"stems": [
  {
    "english": "be black (of horses)",
    "salish": "q'ʷəd(u-)",
    "stem_category": {
      "value": "verbs"
    }
  },
  {
    "english": "dive from land (face in horseshoe curve)",
    "salish": "us-əlš",
    "stem_category": {
      "value": "verbs"
    }
  }
]
```

Why is it needed?

Middleware smooths the communication between the front-end and the database, it helps make the development of the front-end easier in ways we will discuss in the next section. It optimizes the process of asking for and receiving information from the database and making it presentable to users.

Which systems did we choose?

As noted above, our middleware is Hasura. Hasura provides an application programming interface, or 'API', that runs between the Postgres database and the React.js front-end. Which we discuss in the next section.

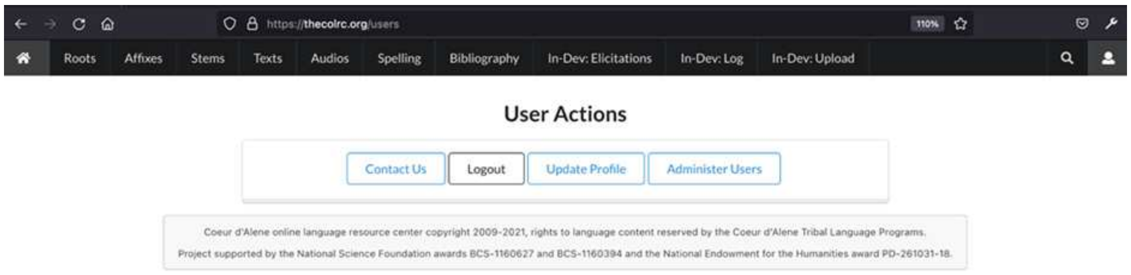
A front-end

The front-end of a web application is everything you actually see, click on, or otherwise interact with when you visit the application online. The job of a front-end is to make it as easy as possible for your users to find what they need, and navigate around all the different resources on your application. Modern front-ends are designed to be 'responsive', meaning that they can change their appearance depending on the size and shape of your screen - changing to fit the screen of a phone, or a tablet, or a desktop or laptop computer.

What does it do?

An application front-end is responsible for all presentations of information to and interactions with website visitors. The front-end's colors, fonts, button styles and other visual elements are managed by scripts called 'cascading style sheets' or CSS. Using CSS allows the developer to create a set of styles and patterns that are applied to all of the elements in an application, or to any subset of elements, without having to include that information in the code that produces the pages themselves. For example, compare these two screen captures from the COLRC 2.0: the user actions display (18), and the update profile display (19):

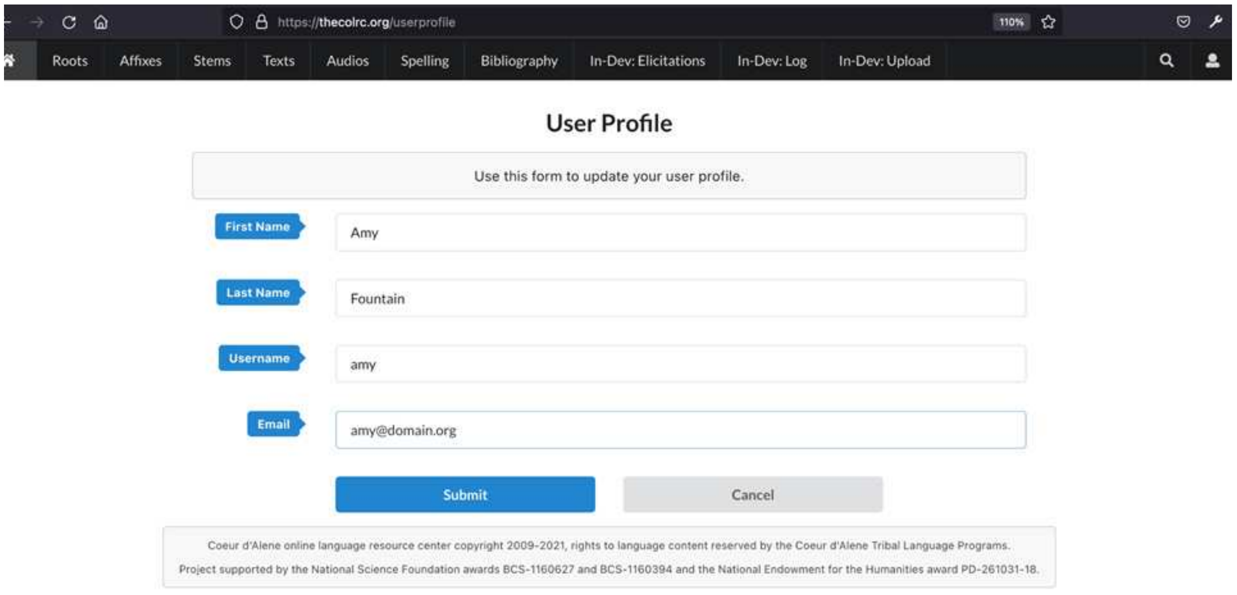
(18) A view of the front-end of our application showing User Actions



Coeur d'Alene online language resource center copyright 2009-2021, rights to language content reserved by the Coeur d'Alene Tribal Language Programs.

Project supported by the National Science Foundation awards BCS-1160627 and BCS-1160394 and the National Endowment for the Humanities award PD-261031-18.

(19) A view of the front-end of our application showing the User Profile update form



User Profile

Use this form to update your user profile.

First Name: Amy

Last Name: Fountain

Username: amy

Email: amy@domain.org

Submit Cancel

Coeur d'Alene online language resource center copyright 2009-2021, rights to language content reserved by the Coeur d'Alene Tribal Language Programs.
Project supported by the National Science Foundation awards BCS-1160627 and BCS-1160394 and the National Endowment for the Humanities award PD-261031-18.

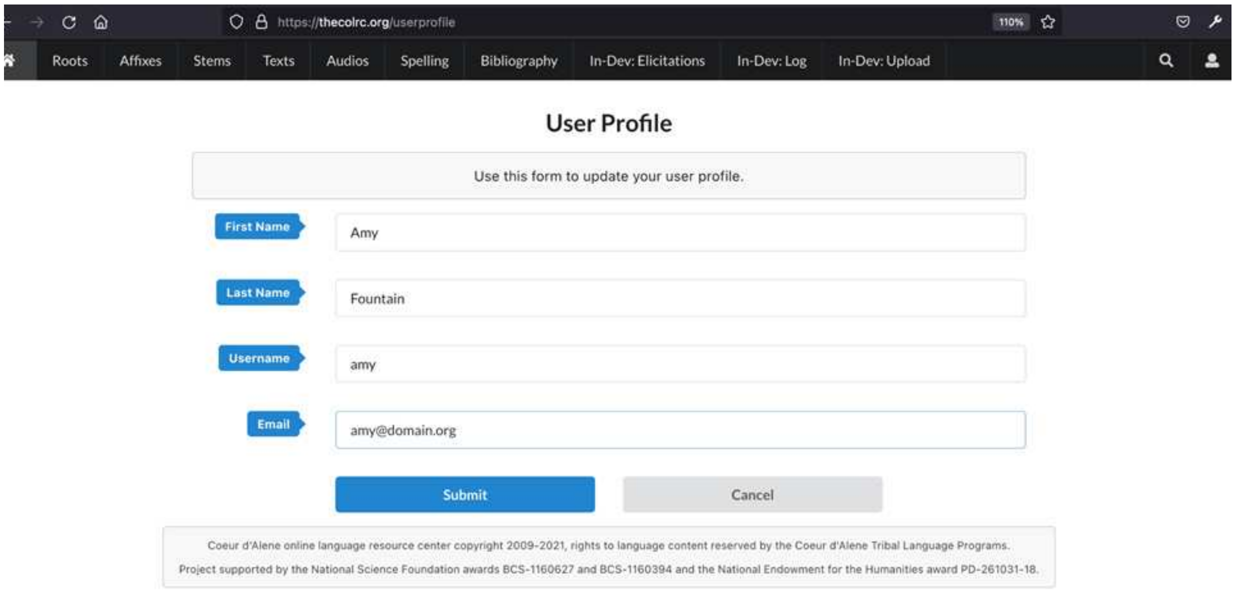
Each of the components displayed above has the same color scheme, the same fonts, and the same overall design. This is because the application uses CSS to control the appearance of each component.

The guts of the front-end of a web application are typically JavaScript. JavaScript is a computer language that runs in internet browser, who read it and convert it to HTML. You can view the HTML that your browser has built by using the 'developer tools' in your web browser, and looking at a pane called 'inspector'. This tool also shows what CSS code is being interpreted in the page, as illustrated in right-hand panel in (20) below:

This component is a set of buttons, each of which is responsible for a given action. Some of these actions go to another component (like 'contact us', which goes to an email interface), some pull data up from the database to allow it to be edited (like 'update profile' and 'administer users'), and some trigger actions that change the session status (like 'logout').

The front-end code produces the text you see, and the buttons, and it also includes functions that do the work of each button (as 'onClick' events). Below is the code that renders the 'User Actions' component:

(20) A view of the 'User Actions' page with 'inspector' showing the rendered html code



User Profile

Use this form to update your user profile.

First Name	Amy
Last Name	Fountain
Username	amy
Email	amy@domain.org

Coeur d'Alene online language resource center copyright 2009-2021, rights to language content reserved by the Coeur d'Alene Tribal Language Programs.
Project supported by the National Science Foundation awards BCS-1160627 and BCS-1160394 and the National Endowment for the Humanities award PD-261031-18.

This code is created in a JavaScript library called 'React.js' (https://react.dev/) which is one of a number of different frameworks that JavaScript can be created within[19]. Let's look at the way the code works.

At the top of the file, we tell the system what other components we are relying on in order to make this page by using 'import' statements. This allows the computer to find the specific code and use it here without us having to retype everything. The component relies on the React.js library, and it uses a CSS system called 'SemanticUI' (https://react.semantic-ui.com/), it also needs to be able to work with our authentication system (which defines a function called "useAuth"), and to find and use a success message ("broadcastSuccess"), some cool externally defined logical functions that allow us to check a users' roles against the roles that are allowed to access each button (from a library called "lodash"), and some permissions logic that we've encoded in a central location ("path_button_permissions").

Below the import statements the component "Users" is declared. It takes an argument called "props", which is information that is shared between components as a user navigates from one component to another. This function defines and builds everything you see in this component, and then 'exports' itself with the last statement in the file "export default Users".

Each 'page' that you see in a web application will be defined by a function like this. When you navigate through an application, for example by clicking a navigation menu button, or by clicking a button in the component, or even by using the 'back' or 'forward' buttons on your browser, you're not really moving through a file system like you do in a static website (as illustrated in (21) above). Instead, you're actually telling the browser to grab and execute a new function and display the result on your screen. The application itself has a central routing function that tells the browser what route it should create in the URL bar (for example, when you see 'https://thecolrc.org/users' as you view the 'Users' component, that's because we've given the application this instruction in its central router. The instruction 'Route' has a path, which is what prints in the URL bar, and it has a component, which is a JavaScript function like the one above, and a key, which is a unique identifier for that component that can be referred to in other components.

[19] It is also possible to just create generic, also referred to as 'vanilla', JavaScript. Libraries like React.js are developed because they help simplify and automate many of the kinds of things that developers have to do over and over again. Interested readers might want to work with Next.js (https://nextjs.org/) which seems to be overtaking 'plain' React as the most popular library at time of writing. These things change very rapidly.

Why is it needed?

A front-end is the way that people will view and interact with your application, so a clean, clear, and functional front-end is crucial to the system's usability. JavaScript is your internet browser's first language, so writing the front-end in JavaScript ensures that your application will work well for users.

Which systems did we choose?

As described above, we chose React.js because it was one of the most commonly used frameworks for creating web applications, and that ensures the durability of that system - as much as any such systems can be described as 'durable'.

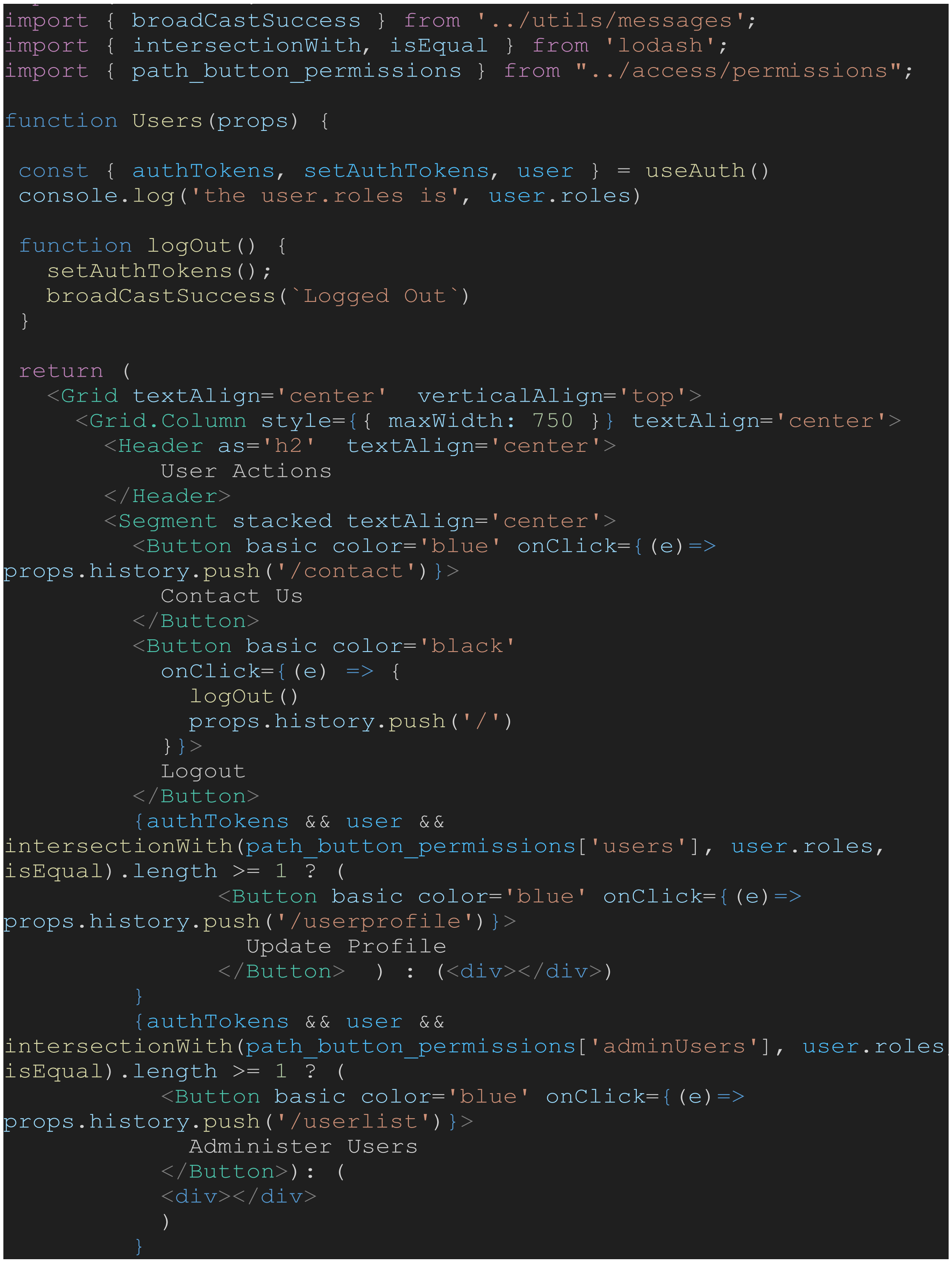
Containerization

Containerization is a system for packing up all the pieces and parts of your application, along with everything it needs to run, into a single 'suitcase' - a unit that can be easily removed from one server and added to another, with no or minimal changes needed to the server itself (Vincent & Fountain 2019b). Further, containerized applications can be designed to run in a self-contained way that does not necessarily require 'superuser' or 'root' access to the operating system of the computer that they run on. Rootless containerization means that not only will your application be easily moveable, but also that it will be more likely to find a good home because it does not pose a security threat to the server where it lives.

Which systems did we choose?

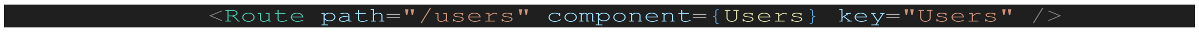
We utilize Docker (https://www.docker.com/) for the containerization of the COLRC. Currently, our docker implementation does require root access to the server where our system lives, but we are in the process of moving to a rootless architecture so that we will have an easier time finding homes in the future.

21. Code that renders the 'User Actions' component of our application.



```
import { broadCastSuccess } from '../utils/messages';
import { intersectionWith, isEqual } from 'lodash';
import { path_button_permissions } from "../access/permissions";

function Users(props) {

 const { authTokens, setAuthTokens, user } = useAuth()
 console.log('the user.roles is', user.roles)

 function logOut() {
   setAuthTokens();
   broadCastSuccess(`Logged Out`)
 }

 return (
   <Grid textAlign='center'  verticalAlign='top'>
     <Grid.Column style={{ maxWidth: 750 }} textAlign='center'>
       <Header as='h2'  textAlign='center'>
           User Actions
       </Header>
       <Segment stacked textAlign='center'>
         <Button basic color='blue' onClick={(e)=>
props.history.push('/contact')}>
           Contact Us
         </Button>
         <Button basic color='black'
           onClick={(e) => {
             logOut()
             props.history.push('/')
           }}>
           Logout
         </Button>
         {authTokens && user &&
intersectionWith(path_button_permissions['users'], user.roles,
isEqual).length >= 1 ? (
               <Button basic color='blue' onClick={(e)=>
props.history.push('/userprofile')}>
                 Update Profile
               </Button>  ) : (<div></div>)
         }
         {authTokens && user &&
intersectionWith(path_button_permissions['adminUsers'], user.roles
isEqual).length >= 1 ? (
           <Button basic color='blue' onClick={(e)=>
props.history.push('/userlist')}>
             Administer Users
           </Button>): (
           <div></div>
           )
         }
```

(22) A path instruction in React.js



```
<Route path="/users" component={Users} key="Users" />
```

6. Conclusion

We hope that we've made it clear that the application is built in ways that are necessary for it to be useful to people, and as durable as software can be. But we also hope that you can see why we can't in good faith refer to the COLRC as an 'archive'.

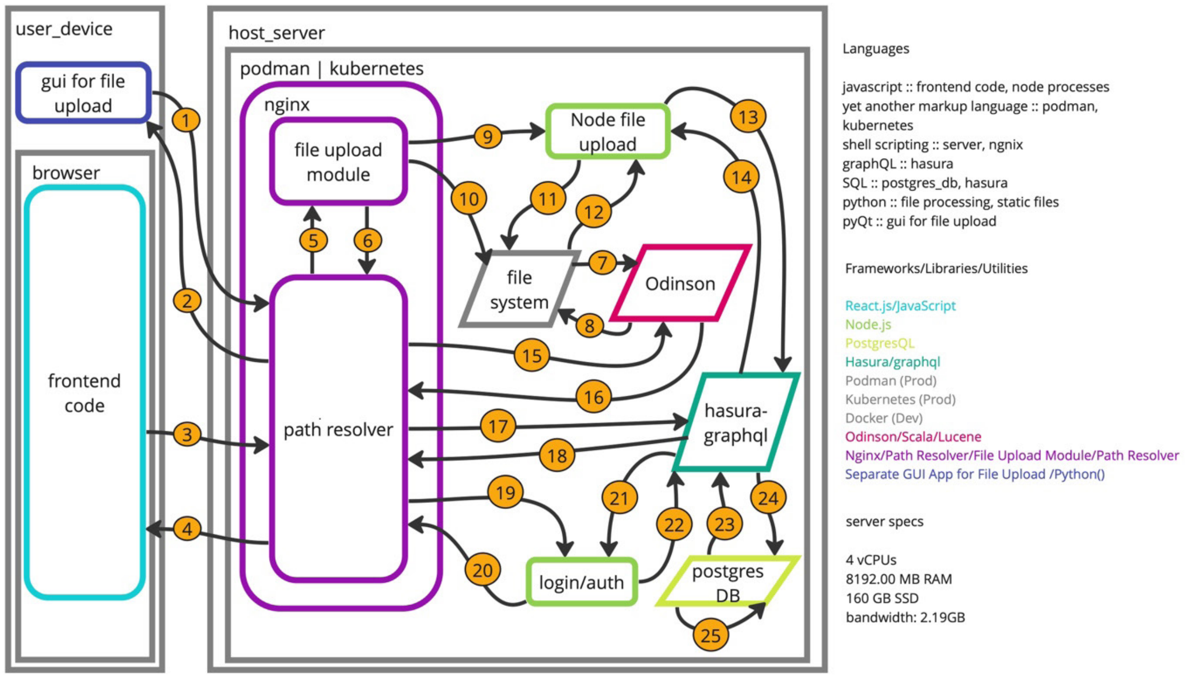
Archives must, above all, provide safe storage for materials in the medium and long term. We've found that guaranteeing long-term stability and safety online is impossible for reasons that are not under the control of any of the participants in our team, and that are not at all unique to our project.

It'll never be done

We hope that it is also clear from the above discussion that technological systems like the COLRC can never really be 'done', because of the very short half-life of servers, software, and file types. Electronic resources, like physical resources, require ongoing maintenance and care - and much more than physical resources, constant (re-)creation. That is, it is an ever-unfinished research product that involves taking in new information, digitizing old materials, and navigating developments in digital infrastructures, formats, and standards (Berez-Kroeker 2016: 426))

The work of the COLRC will never be done, and it will never stop requiring support. Luckily, the work required is interesting and beneficial, and the community supports it. We cannot ask for more.

Appendix A: Complete technical schematic for all components of the COLRC 2.0



References

Abba, Ihechikara Vincent. "What is an ORM – The Meaning of Object Relational Mapping Database Tools." freeCodeCamp.org. 2022. https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/.

Benedicto, Elena. "When Participatory Action Research (PAR) and (Western) Academic Institutional Policies Do Not Align." In Insights from Practices in Community-Based Research: From Theory to Practice Around the Globe, edited by Shannon Bischoff and Carmen Jany, 38-65. 2018.

Bird, Steven, and Gary Simons. "Seven Dimensions of Portability for Language Documentation and Description." Language 79 (2003a): 557–582.

Bird, Steven, and Gary Simons. "Extending Dublin Core Metadata to Support the Description and Discovery of Language Resources." Computers and the Humanities 37 (2003b): 375–388.

Bischoff, Shannon T. Lynx: A Morphological Analysis and Translation of Dorothy Nicodemus' Coeur d'Alene Narrative. Missoula: University of Montana MA thesis, 2001.

Bischoff, Shannon T. Functional Forms-Formal Functions: An Account of Coeur d'Alene Clause Structure. Tucson: University of Arizona Ph.D. dissertation, 2007.

Bischoff, Shannon T. "Lexical Affixes, Incorporation, and Conflation: The Case of Coeur d'Alene." Studia Linguistica 65 (2011a): 1–31.

Bischoff, Shannon T. Formal Notes on Coeur d'Alene Clause Structure. Cambridge Scholars, 2011b.

Bischoff, Shannon, and Amy Fountain. "A Case-Study in Grass Roots Development of Web Resources for Language Workers." In The Persistence of Language: Constructing and Confronting the Past and Present in the Voices of Jane H. Hill, edited by Shannon Bischoff et al., 175. John Benjamins Publishing, 2013.

Bischoff, Shannon, Amy Fountain, and Audra Vincent. "100 Years of Analyzing Coeur d'Alene with the Community." In Insights from Practices in Community-Based Research: From Theory to Practice Around the Globe, edited by Shannon Bischoff and Carmen Jany, 194-211. 2018.

Bischoff, Shannon T., and Carmen Jany, eds. Insights from Practices in Community-Based Research. De Gruyter Mouton, 2018.

Byron, Lee. "GraphQL: A Data Query Language." Engineering at Meta. 2015. https://engineering.fb.com/2015/09/14/core-data/graphql-a-data-query-language/.

Chang, Debbie. TAPS: Checklist for Responsible Archiving of Digital Language Resources. Dallas, TX: Graduate Institute of Applied Linguistics, Dallas International University, 2010.

Chang, Debbie. TAPS: Checklist for Responsible Archiving of Digital Language Resources. Dallas, TX: Graduate Institute of Applied Linguistics MA thesis, 2010.

Codd, Edgar F. "A Relational Model of Data for Large Shared Data Banks." Communications of the ACM 13, no. 6 (1970). https://doi.org/10.1145/362384.362685.

Date, C. J. A Guide to DB2: A User's Guide to the IBM Product IBM Database 2, a Relational Database Management System for the MVS Environment and Its Companion Products QMF and DXT. Reading, Mass.: Addison-Wesley Pub. Co., 1984. http://archive.org/details/guidetodb2users00date.

Doak, Ivy G. "Another Look at Coeur d'Alene Harmony." International Journal of American Linguistics 58 (1992): 1–35.

Doak, Ivy G. Coeur d'Alene Grammatical Relations. Austin, TX: University of Texas dissertation, 1997.

Doak, Ivy G., and Anthony Mattina. "Okanagan-lx, Coeur d'Alene-ilš, and Cognate Forms." International Journal of American Linguistics 63 (1997): 334–361.

Doak, Ivy G., and Timothy Montler. "Orthography, Lexicography and Language Change." In Endangered Languages and Literacy. Proceedings of the Fourth FEL Conference, edited by Nicholas Ostler and Blair Rudes. 2000. http://montler.net/papers/OrthographyFEL22000.pdf.

GQL Standards Committee. "Graph Query Language GQL." https://www.gqlstandards.org/.

Henke, Ryan E., and Andrea L. Berez-Kroeker. "A Brief History of Archiving Language Documentation, with an Annotated Bibliography." Language Documentation & Conservation 10 (2016): 411-455.

Johnson, Heidi. "Language Documentation and Archiving, or How to Build a Better Corpus." Language Documentation and Description 2 (2014): 140-153.

Junker, Marie-Odile. "Participatory Action Research for Indigenous Linguistics in the Digital Age." In Insights from Practices in Community-Based Research: From Theory to Practice Around the Globe, edited by Shannon Bischoff and Carmen Jany, 164-175. 2018.

Lyon, John, and Rebecca Greene-Wood, eds. Lawrence Nicodemus' Coeur d'Alene Dictionary in Root Format. Missoula, MT: University of Montana Occasional Papers in Linguistics, 2007.

Nicodemus, Lawrence G. "The Coeur d'Alene Language Project." ICSL 8 (1973). Eugene, OR.

Nicodemus, Lawrence G. Snchitsu'umshtsn: The Coeur d'Alene Language. A Modern Course. Plummer, ID: Coeur d'Alene Tribe, 1975a.

Nicodemus, Lawrence G. The Coeur d'Alene Language in Two Volumes: I The Grammar and Coeur d'Alene-English Dictionary; II English-Coeur d'Alene Dictionary. Spokane, WA: University Press, 1975b.

Nicodemus, Lawrence G., Wanda Matt, Reva Hess, Gary Sobbing, Jill Maria Wagner, and Dianne Allen. Snchitsu'umshtsn: Coeur d'Alene Reference Book Volume 1. Plummer, ID: Coeur d'Alene Tribe, 2000a.

Nicodemus, Lawrence G., Wanda Matt, Reva Hess, Gary Sobbing, Jill Maria Wagner, and Dianne Allen. Snchitsu'umshtsn: Coeur d'Alene Reference Book Volume 2. Plummer: Coeur d'Alene Tribe, 2000b.

Reichard, Gladys Amanda. "Coeur d'Alene." In Handbook of American Indian Languages Part 3, edited by Franz Boas, 515–707. New York: J. J. Augustin, Inc., 1938.

Reichard, Gladys Amanda. "Stem-List of the Coeur d'Alene Language." International Journal of American Linguistics 10 (1939): 92–108.

Reichard, Gladys Amanda, and Adele Froelich. An Analysis of Coeur d'Alene Indian Myths. Philadelphia: Memoirs of the American Folk-lore Society, v. 41, 1947.

Vincent, Audra, Shannon Bischoff, and Amy V. Fountain. "Tgwe'l Nok'o'qin He Spintch 'Itsmeyptsni'wes Hiɫ 'Itsqhwaq'wpmi'wes 'Ul Snchitsu'umshtsn: One Hundred Years of Learning and Analyzing the Coeur d'Alene Language Together." In Washington D.C., 2016.

Vincent, Audra, and Amy V. Fountain. "Developing a Language Archive." Workshop presented at the Symposium for American Indian Languages, University of Arizona. 2019a. https://sites.google.com/email.arizona.edu/sail2019-developing-an-archive/.

Vincent, Audra, and Amy V. Fountain. "Pute'nt Khwa Isqwa'qwe'el Honor Your Language." Workshop, collaborative with Coeur d'Alene Language Programs and American Indian Language Development Institute presented at the Pute'nt Khwa Isqwa'qwe'el Honor Your Language, Whorley, ID. 2019b. https://tinyurl.com/honoryourlanguage.

Yi, Irene, Amelia Lake, Juhyae Kim, Kassandra Haakman, Jeremiah Jewell, Sarah Babinski, and Claire Bowern. "Accessibility, Discoverability, and Functionality: An Audit of and Recommendations for Digital Language Archives." *Journal of Open Humanities Data* 8 (2022): 1–19. https://doi.org/10.5334/johd.59.